Deanonymization of Tor Onion Services with Acceleration-Based Watermarking

(extended abstract of the MSc dissertation)

Afonso Manuel Vieira Machado Barros de Carvalho
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisors: Professor Nuno Santos
Professor Diogo Barradas

***Abstract*—The Tor anonymity network and its Onion Service (OS) infrastructure allow users to browse and provide services on the Internet while benefiting from sender and receiver anonymity. This enables them to circumvent censorship filters and remain safe from prosecution but also provides a means to conduct illegal activities. This has made Tor an enticing target for attackers, including Law Enforcement Agencies (LEAs) seeking to identify unlawful OSes. These LEAs may form coalitions, gaining broad access to network traffic information collected at the level of Autonomous Systems or Internet Exchange Points. This information is known to allow deanonymization attacks on Tor's infrastructure to take place.**

In fact, recent attacks have shown that it is possible for a global passive adversary to fully deanonymize Tor Onion Sessions with high accuracy. The proposed techniques, however, depend on the monitoring of a large amount of Tor flow samples introducing serious bottlenecks to the scalability of these attacks and to the maximum achievable recall (i.e., the number of sessions that can effectively be screened by the adversary). In this work, we propose an alternative attack, DissecTor, that utilizes a new acceleration-based watermarking scheme and machine learning techniques to deanonymize a targeted OS both effective and covertly. By identifying the target OS one significantly reduces the complexity and resources needed to fully deanonimize the whole browsing session. We performed an in depth evaluation of the DissecTor system, exploring several variants and reflecting on the best use cases for each.

I. Introduction

This work focuses on the usage of traffic correlation attacks to break the anonymity guarantees of the Tor network, specially its Onion Service (OS) infrastructure.

Tor is a popular anonymity network whose privacy guarantees have made it a fundamental tool for promoting free speech and protecting the privacy of Internet users. In fact, Tor has helped a vast range of users – ranging from journalists and political activists to whistleblowers and citizens living in repressive regimes – to browse the Internet privately and enabling such users to circumvent censorship filters and to remain safe from prosecution [1].

Tor is built on top of the onion routing protocol [2], whereby a message is encapsulated in successive layers of encryption and transmitted along several relay nodes before arriving at its final destination. Each node that receives the message is able to decipher the top-most layer of encryption in order to uncover the data's next destination. Only the last node can decipher the message's contents. This means that each intermediate node in the chain, called a circuit, knows only the identity of its predecessor and successor, keeping the sender's anonymity intact.

In Tor, this principle is not limited to clients. Tor's Onion Services (OSes), previously called Hidden Services (HSes), are websites or services that use the Tor technology to keep their identities (i.e., IP addresses) secret. Unlike "normal" websites, where the client knows the service's IP and the message is directly sent to it, an OS's address is kept hidden from the client. A rendezvous node is used as intermediary for the message exchange and both the client and OS communicate with it using onion routing in the form of two separate Tor circuits.

Due to its strong anonymity properties, Tor has since become an interesting target for state-level actors. For instance, repressive governments are interested in breaking Tor to prevent abuse and political dissidence [3], while law enforcement wish to fight organized online crime [4]. Many attacks targeting Tor have thus been proposed. These attacks can be classified as *passive* or *active*, if the attacker only observes traffic or has the ability to manipulate it, respectively, and as *single-end* or *end-to-end*, if the attacker monitors/controls only one of the edge relays of a Tor circuit or both simultaneously.

One of the most significant threats to Tor is that of correlation attacks [5], [6], end-to-end attacks where an adversary monitors both edges of a Tor circuit and seeks to establish a relation between flows observed at the entry and exit points. Since the entry relay knows the sender and the exit relay knows the receiver, sound correlations can effectively deanonymize the communications pertaining to that specific Tor circuit and enable the monitoring of a user's activity. To make matters worse, the success of passive correlation attacks can be significantly improved through the usage of active watermarking techniques [7], [8], [9]. Put briefly, an attacker can carefully manipulate the traffic patterns of specific Tor circuits (e.g., introduce predictable delay on packets) to ease correlation efforts.

Recently a new attack called Torpedo [6] was proposed seeking to enable a global passive adversary to deanonymize Tor onion service sessions, i.e. deanonymize OSes and their users and establish correlation of the latter's browsing sessions. Despite its precision, this attack is seldomly suc-

cessful and achieves low coverage, missing the majority of correlated flows. This is due to it being both very expensive from a computational point of view and poorly scalable.

The goal of this work is to investigate an alternative attack that aims specifically to deanonymize a targeted OS (as opposed to deanonymizing both the client- and the server-side of OS sessions). Assuming that the attacker can control the client-side and generate arbitrary traffic directed towards a predefined OS, we aim to show that it is possible to launch correlation attacks to deanonymize Tor Onion Services with higher coverage and scalability than Torpedo's.

To achieve this goal, the solution we propose leverages active attacks based on watermarking combined with traffic correlation techniques. The central technical challenge lies in finding a watermarking technique that can simultaneously generate a watermark that (a) can be accurately detected at the ingress link of the targeted OS, and (b) is stealthy enough to prevent being detected by the OS provider.

In this work we present a new system called DissecTor able to deanonymize specific Tor Onion Services. DissecTor uses an active approach, relying on flow watermarking and machine learning, in order to reveal the target OS's IP address. We present the following contributions:

- The design and implementation of DissecTor, which can launch active attacks to targeted OSes in two different stages: (1) send watermarked traffic to a target OS, and (2) identify the watermarked connection between the OS and its guard node from a pool of candidates using machine learning techniques.
- The development of a novel watermarking technique named *acceleration-based watermarking* which allows for deanonymizing OSes by identifying predefined variations in the receiver's traffic based on perturbations introduced by probes sent by the attacker.
- An extensive evaluation of our system which covers two different watermark detection techniques and and that discusses how their parameterization might balance the attack's accuracy and stealthiness.

II. Related Work

A. Traffic Correlation Attacks

Traffic correlation attacks, occur when an adversary attempts to link two network flows solely based on the observable features of the traffic and not its contents.

Correlation attacks performed against Tor rely on flows captured at both ends of a circuit and do not need to know its full path. Since each edge relay knows the identity of one of the communicating parties, a successful correlation attack effectively results in the deanonymization of both the user's IP and the IP it is accessing. Correlation attacks can be classified according on the different techniques they employ:

- Timing-based: If they use statistical analysis of flow features such as inter-packet arrival times, packet volume and bursting patterns [10];
- Bandwidth-based: If they focus on perturbation in the available bandwidth towards the target [11];
- Application-based: If they are based on the interaction between the target and an attacker-controlled application that induces known traffic features [12];
- Watermark-based: If the attacker manipulates traffic near its source seeking to induce an identifiable signature towards the target [7], [9].

State-of-the-art traffic correlation attacks leverage deep learning techniques whether they assume a passive [5], [6] or active [9] approach. This allows them to better cope with the Tor network's characteristics, boosting their success.

B. Watermarking Techniques

Network flow watermarking is a type of traffic analysis where a "small piece of information that can be used to uniquely identify a connection" [13], the watermark, is embedded into flows near their source, allowing for their identification at another point of the network [14].

The technique of traffic analysis with flow watermarking may be divided in two phases: converting information into a watermark, and embedding it into the flow – carried out by the *watermarker* – and observing a flow, identifying it as watermarked and decoding and retrieving the information from the watermark – carried out by a *watermark detector*. According to Iacovazzi et al. [14], watermarking systems may be classified in respect to: i) Diversity scheme; ii) Carrier; iii) Blindness, in what concerns the detection system.

Diversity schemes describe how a signal spreads in specific domains. There are three classes of diversity schemes: *time*, *frequency*, and *space*. Time diversity is based on replicating a pattern over time in a recognizable way [7], [15]. This can be done, for instance, by inducing controlled delays in a flow's packets. Frequency diversity uses manipulation of a flow's rate to transmit the watermark [16]. Space diversity uses multiple flows similarly watermarked in order to transmit a single signal. This is done so that a lightweight, difficult to detect, signal is amplified for detection [17].

The Carrier is the traffic feature where the watermark will be embedded. Carriers can be *content*, *timing*, *size* and *rate-based* [14]. Content-based watermarking consists in embedding the watermark directly in the contents of the messages [13]. Timing-based approaches carry the signal in the sequence of arrival and/or departure times of packets measured at a given point in the network [7], [15]. Size-based techniques usually resort to altering flow packet lengths [18]. Finally, rate-based watermarking carriers consist of fluctuations imposed on the real traffic rate by the injection of dummy traffic. [16].

Watermark detection depends on the encoding chosen for the watermark as well as its diversity scheme and carrier. The watermark detector extracts relevant features from sniffed flows using them to determine the presence or content of the watermarks. Watermark decoding algorithms may be *non-blind* [7], [15] or *blind* [19] regarding, respectively, their dependence or not on carrier data provided by the watermarker.

Table II-B shows how different watermarking systems proposed in the literature are classified and how DissecTor,

Table I
HOW DISSECTOR COMPARES TO SOME OF THE PROPOSED WATERMARKING SYSTEMS.

System	Diversity Scheme	Carrier	Blindness
Wang et al. [19]	Time	Time-based	Blind
Rainbow [7]	Time	Time-based	Non-blind
Swirl [15]	Time	Time-based	Non-blind
Inflow [8]	Time	Time-based	Non-blind
Duster [20]	Frequency	Rate-based	Blind
Finn [9]	Time	Time-based	Blind
DissecTor	**Frequency/Space**	**Rate-based**	**Non-blind**

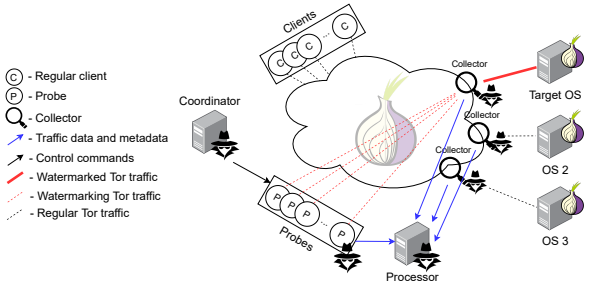

Figure 1. DissecTor's architecture and main component interactions

the system we conceived, fits into the classification system covered until now.

III. DISSECTOR

A. Overview and Pipeline

DissecTor, illustrated in Figure 1, is a distributed system conceived to allow digital crime investigators to target specific Onion Services (OSes) and reveal their location in the form of IP addresses. For DissecTor to work, it is necessary to have access to a large number of network vantage points where traffic data can be collected. DissecTor assumes these vantage points are located at the Autonomous System (AS) level. This access is materialized in the form of modules, named *collectors*, that are to be installed throughout the network and whose job is to collect traffic data between the OS and its guard node in the Tor circuit being used.

Other than the already mentioned collectors, the system is made out of three more types of components: a *coordinator*; *probes* and *processor(s)*. Figure 1 shows how these components interact with each other and with the Tor network to perform an attack. In particular, the figure shows how the coordinator is responsible for issuing commands. It does so to both the collectors and the probes even though the arrows connecting the coordinator to the collectors were omitted from the figure as to not hinder its clarity. This component has no direct interaction with the Tor network or the target. We can also see that a varying number of probes is involved in the attack. This number is determined by the user and will directly influence the watermark signature. The probes interact with the network as if they were regular clients. The only difference between a probe and a regular client is related to the timing and number of requests sent towards the OS, as will become clear further in this section. Furthermore, it is also shown how the probes (each one separately) relay traffic information to the processor, as do the collectors. The processor, similarly to the coordinator, does not interact with the Tor network or the target and simply receives traffic information from the probes and the collectors. The remainder of its functionality is performed offline. The processor is responsible for watermark detection and for returning the attack's result. It is also possible to distribute the processing workload over several instances of the processor component although we do not represent said scenario in the figure.

Investigators are able to launch an attack on a specific target OS via the system's user interface. To best understand DissecTor's pipeline let us examine the example of a law enforcement agent that seeks to uncover the IP address of an OS that serves as an online narcotics marketplace. To begin the attack, the agent must input the desired runtime parameters. These consist of: the onion address of the target x; the number y of probes to use during the attack which may or may not be based of prior knowledge about the popularity of the OS; a set of probing times T which must have one or more timestamps of when the probing sessions are to take place and, finally, a confidence threshold z. In the example of an attack on the narcotics marketplace, the agent might input *verycheapdrugs.onion* as variable x, 10 as y, 3 different times the next day as T and 80% as z.

Once the runtime parameters are collected, the coordinator, which is responsible for the synchronization of all other system components and the overall orchestration of the attack, launches and then configures 10 probe instances. A predetermined amount of time before the first probing session, at 11am, the coordinator signals all collectors to start recording traffic data.

At 11am, when the first probing session is to take place, all 10 probes concurrently send requests to *verycheapdrugs.onion* such that a traffic pattern, hereby referred to as a *watermark*, is directly embedded in the ingress and indirectly embedded in the egress traffic connecting it to its guard node. During the session all probes are responsible for collecting metadata, namely timestamps and volume of all network layer packets, which will later be used to guide the watermark detection. Once the 11am session in concluded, the system stays idle until the next probing time, 1pm, arrives and the process is repeated.

Once all probing sessions take place the coordinator signals the collectors to stop recording traffic data. The data, both from the collectors and the probes, is sent to the processor which is responsible for *watermark detection*. If the confidence in the obtained results exceeds 80% these are then presented to the investigator.

B. Acceleration-Based Watermark

DissecTor employs a new watermarking technique – named *acceleration-based watermark* – based on induced traffic patterns detectable on the connection between the target OS and its guard node. These patterns are manifested in the form of throughput spikes in both the ingress and egress flows of said connection. For this reason, we classify our watermark as taking advantage of a rate-based carrier – its diversity scheme can be seen as combination of frequency

and space schemes. Finally, the watermark detection is non-blind, relying on data provided by the watermarker (which in our system consists of the set of probes).

To embed the watermark, each probe, acting as a regular client, fetches the target's landing page while strictly registering the departure and arrival times of both the request's and response's network packets. This request, which is not different from any other the OS might receive, is directly reflected on traffic flowing between the guard and the OS in the form of an increase in the data-rate followed by its decrease to previously existing levels. These spikes can be located within the time window delimited by the departure time of the request and the arrival time of the response recorded by the probe. If we separate traffic according to its direction we find there are two distinct spikes, respectively in the ingress and egress flows, the former being induced by the request and the latter by the response. These spikes differ from one another and can both be used in conjunction when performing watermark detection.

To perform the detection, the processor begins by dividing the data provided by the collectors according to the number of possible *marks*. A mark, identified by its IP address, is a possible match for the target OS. The number of *streams* (defined next) associated to each mark is dependant on the watermark detection method used further down the pipeline and ranges from 1 to 3. Each stream is a collection of packet sizes and timestamps which the processor then groups into time intervals of granularity G, in seconds, for easier processing. DissecTor may use 3 possible streams: the one containing packets sent to the mark, named *fetch stream*; the one containing packets sent by the mark, named *reply stream* and one that contains all packets, named *joint stream*.

Comparing each mark's streams is the next logical step in the process. Our goal is to detect DissecTor's induced spikes (the watermark) on the set of candidate streams. This requires comparing each stream's fluctuations of throughput, given in B/s, leading us to choose the *acceleration*, given in B/s^2, as the natural metric to use in our analysis. Using the acceleration is also adequate when comparing sets of different candidate flows whose baseline/average throughput vary widely since the induced variation is independent of those and only relates to the OS's response size, time and the number of requests sent by the probes. Each stream's packet size and timing data is used to calculate the throughput (see Equation 1) in intervals Δt of size G, where Δs is the total amount of bytes transmitted during said interval.

$$throughput = \frac{\Delta s}{\Delta t} \tag{1}$$

$$acceleration = \frac{\Delta throughput}{\Delta t} \tag{2}$$

These throughput values can then be used to calculate the acceleration (see Equation 2) between consecutive samples.

After calculating a vector of accelerations (see Figure 2), where each value corresponds to an interval of size G for a given flow, these are grouped into *buckets*. The size of each bucket depends on the duration of the probing sessions and is calculated according to the metadata collected by the probes. Every acceleration value corresponding to packets observed during a probing session is grouped into the same bucket. For example, in the case of Figure 2, all values between the orange lines will belong to the same bucket. The remaining acceleration values in the vector do not correspond to the watermark and are grouped into buckets either by duration B (as many B seconds sized buckets as possible) or by divisor D (grouped into D equally sized buckets). Although there are several ways of forming the buckets, once a method is chosen it must be applied to all marks' acceleration vectors. This ensures that the n^{th} bucket of every stream starts and ends exactly at the same timestamp and can, therefore, be compared. Finally, each bucket is then labeled as *"watermark candidate"* or *"normal traffic"* according to when it occurred and a set of summary statistics is calculated over each one. The resultant datasets will be the basis to perform the watermark detection. It is important to note that the used statistics do not depend on the amount of acceleration samples per bucket ensuring that the "watermark candidate" buckets do not stand out from the others solely due to their duration.

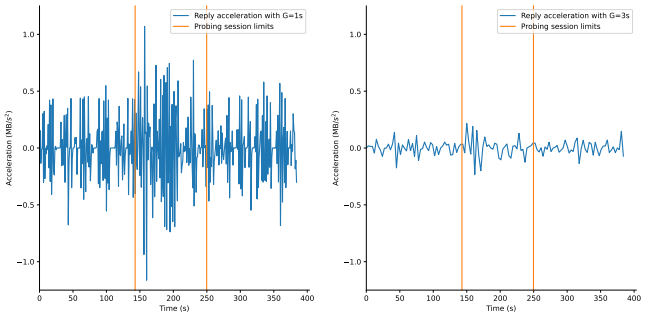

Figure 2. Difference in calculating the same stream's acceleration with G=1s (left) and G=3s (right). A probing session took place in the period of time between the orange vertical lines.

C. Watermark Amplification

Because at any given time there might a large number of concurrent clients browsing the target OS, and since every request is responsible for its own variation in transmission rate the necessity arises to make DissecTor's variation, to which we've been referring as a spike, more identifiable. This should be done without raising suspicion from the OS.

The chosen approach was to make the probe module act as if it were a normal client sending a fetch request to the OS's landing page but have several different probes do so simultaneously in order to amplify the strength of our watermark. This is the reason our watermarking scheme may be referred to as space-based.

This approach presents its own set of challenges since an insufficient number of probes would mean the watermark remained undetectable, being lost amidst the OS's expected baseline traffic rate but an excessively large set of probes could make the attack too obvious.

A different challenge is posed by the fact that each probe, being independent, is expected to establish its own Tor circuit to the target OS. This results in different Round-trip Times (RTT) for each probe and a significant practical difficulty in ensuring that different probes' requests will arrive at the OS simultaneously. Two possible ways of dealing with this problem are i) to increase the number of successive requests each probe sends in a probing session and ii) to decrease the granularity used by the processor when calculating the accelerations (see Figure 2). The former increases the likelihood of the requests overlapping and the latter allows the impact of several non-simultaneous but closely timed requests to be grouped in the same calculation. Nonetheless, both approaches have their disadvantages as the first implies longer, less precise and less covert attacks and the second less data points and consequent contamination of watermark data with baseline traffic.

D. Watermark Classifiers – Training and Detection

The first of the two machine learning techniques used to perform watermark detection in DissecTor uses supervised learning classifiers. These models are trained with the datasets mentioned at the end of Section III-B with both watermarked and unwatermarked samples and a set of features derived from a set of summary statistics. This statistical approach to feature selection is similar the one employed in the encrypted traffic fingerprinting literature [21].

Once the models are adequately trained we use them to classify the buckets marked as "watermark candidates". Since several classification models exist it is essential to choose the best one for the task at hand. We evaluated several of them and present the results in Section IV.

The biggest drawback of this approach resides in the necessity for training the classifier. Although this can be done by setting up mock scenarios where realistic conditions are simulated to collect data, this is specially difficult since privileged information regarding the target is required. Given said difficulties, this approach poses obvious obstacles to investigators looking to employ it. A different technique, that does not rely on privileged information, is therefore much more suited to this application.

E. Anomaly Detection

We conducted a set of experiments that rely on anomaly detection techniques instead of supervised classifiers to identify the watermarked buckets by their statistical differences from unwatermarked ones. The methodology we employed is similar to Zhang et al.'s [22] and relies on One Class Support Vector Machines (OCSVMs) [23] as an adaptation of Support Vector Machines (SVMs) to one class classification problems (as is the case of determining the presence or absence of our watermark).

Contrary to the previously covered classifiers, where a model is trained with samples from all classes, using OCSVMs only requires data points belonging to one of them. As Zhang et al. [22] explained, this data is then mapped into a multidimensional space based on its features after which a decision function is derived. This function is afterwards used to determine if new data points belong to the class used in the training stage or if they are anomalies, therefore belonging to the *other* class.

This methodology eliminates the need to construct mock scenarios in order to collect samples of *both* watermarked and unwatermarked buckets. Using OCSVMs for anomaly detection requires only unwatermarked traffic as training for the model. Since this traffic collection process can be done immediately before and/or after a probing session takes place, we can ensure the OCSVM is trained with the most realistic data by simply adjusting when to start and stop the collectors at the time of the attack.

After training the OCSVM with data from a specific candidate, we present it with the buckets collected from that specific candidate at the time of the probing sessions. Only the target's stream should have its buckets classified as an anomaly, effectively deanonymizing it.

IV. Evaluation

DissecTor's main goal is to perform the deanonymization of a specific Tor Onion Service. This means that our system should be able to accurately identify said OS's IP address from a pool of candidate OSes. In addition, DissecTor should be able to carry out this task in a covert fashion.

Having the system's two goals in mind, DissecTor's evaluation is focused on a) assessing the system's ability to reliably identify a target's stream(s) as being watermarked; b) assessing whether the system has the ability to reliably identify non-target stream(s) as not being watermarked and; c) studying several possible configurations of the system to identify the ones that allow for the best results, and; d) discussing if the configurations that would allow the system to keep a "low-profile" can realistically achieve satisfactory OS deanonymization results.

A. Experimental Testbed

We setup an OS serving a web page approximately 1MB in size to serve as target for DissecTor. This OS was to be as realistic as possible and thus a baseline traffic signature was established over which our system's watermark could be applied. To do so, we arranged for a set of 4 clients that would fetch said page, wait for it to load, sleep a random amount and repeat the process until signalled to stop.

After the baseline was established, we conducted probing experiments where groups of probes would fetch the target's landing page 5 times in a row without intervals (see Section III-C). Once all 5 fetching requests were answered for all participating probes, these would then sleep until the next session. A total of 50 probing sessions was carried out for each set of probes.

All involved parties (probes, clients, OS and coordinator) were deployed on Google Cloud Compute Engine. The clients and the coordinator used machines with 1vCPU and 4GB of RAM and the OS and each probe used machines with 2vCPU and 8GB of RAM.

While the probing experiments took place, traffic data was collected both on the side of the OS and that of the probes simulating the action of the collector components.

In our experiments we used 1, 2, 4, 8 and 12 probes which allowed us to collect data concerning Probe-to-Client Ratios (PCR) of, respectively, $\frac{1}{4}$, $\frac{1}{2}$, $1, 2$ and 3. In the processing stage we used $G = 1s$ and the buckets that did not correspond to probing sessions were created using the "divisor" strategy for $D = 1$ (see Section III-B). The process was applied to the 3 different possible streams – Reply, Fetch and Joint. In total, and counting each stream separately, we ended up with 1500 buckets, 750 corresponding to probing sessions and 750 to baseline traffic. The 750 buckets corresponding to probing sessions were equally split by the 5 PCRs, at 150 buckets each.

B. Watermark Detection

The evaluation process of DissecTor consisted in a set of experiments where several system configurations were compared. In our experiments, we made use of several detection metrics: i) True Positive (TP) and False Positive (FP) rates, where "positive" samples correspond to buckets that contain the watermark and "negative" samples do not; ii) F1-score [24] and iii) P4 [25].

The F1-score metric is based on both *Precision* and *Recall*, which are indicative of a classifier's performance regarding "positive" samples and classifications. Being solely based on precision and recall means the F1-score can, to some extent, neglect "negative"-related performance. Given that DissecTor must not only be capable of identifying the watermark but also of not flagging innocent marks as the target, we decided to complemented our analysis with the P4 metric, which was recently proposed with the objective of tackling F1-score's shortcomings. P4 is based on both *Precision* and *Recall* and their "negative" counterparts: *Negative Predictive Value (NPV)* and *Specificity*.

C. Watermark Detection via Supervised Learning

As a benchmark for the classifier-based version of the system we used Weka-v3.8.6's [26] implementation of the C4.5 decision tree algorithm [27]. Like all other classifiers we studied, this classifier was run with default parameters, cross validation with 10 folds and was trained with 250 samples (buckets) of baseline traffic and 50 of watermarked traffic. We did so for every stream and PCR (see Section IV-A) totalling 15 executions.

In the results, we observed that the TP rate ranged from a minimum of 76% in the Fetch stream when using $\frac{1}{4}$ PCR to a maximum of 98% for a PCR of 3 independently of the stream. In turn, the FP rate ranged from 4% for the Fetch stream with $\frac{1}{2}$ PCR to 0.4% in all streams using 3 as PCR. Our results also revealed different trends: i) the TP rate increased with greater PCR values ii) the FP rate decreased with greater PCR values and iii) Reply streams offered the highest TP rates and lowest FP rates of the 3, being the most consistent with respect to the previous tendencies.

Furthermore, when analysing the values of both the F1-score and the P4 we saw that the Reply stream also outperformed the others in both metrics for all values of PCR with exception of $PCR = 1$ (where it performed the worst) and $PCR = 3$ (where all streams performed the same).

We consider the overall results of C4.5 satisfactory given the fact that even the lower PCRs obtained TP rates up to 90% and FP rates no greater than 4%. In Section IV-E, we present the extended results of a set of experiments performed with different classifiers towards improving DissecTor's ability to detect watermarks using supervised learning techniques. When doing so, we will make use of these F1-score and P4 results to compare the different variants as well as the different available streams and PCR values.

D. Watermark Detection via Anomaly Detection

In the experiments conducted on the anomaly detection version of the system we used the OCSVM implementation provided by Scikit-learn [28] v1.1.1.

Unlike the case of supervised learning, this technique requires that a model be trained exclusively with one class of data. In the particular case of DissecTor, all models were made to recognize unwatermarked data and treat watermarked samples as anomalies. To this extent we only trained with unwatermarked buckets. This meant training the algorithms once for each parameter configuration and stream instead of 5 times (as the PCR had no impact in the training set). A direct consequence are FP rates that are independent from the amount of probes used.

When running the experiments we had to define a set of parameters that influenced the behaviour of, and the results obtained by, the model. One of such parameters was the "kernel function" which, after preliminary experiments, was set to use Radial Basis Function, or RBF, throughout. Other parameters, however, justified a more thorough study, constituting the basis for the variants evaluated later on:

Gamma (default: "scale"): Defines the weight each training point has on the resulting decision boundary. If the value of "Gamma" is too large the model can incur in overfitting, reporting more anomalies than it should, and if it is too small it may become too permissive, not reporting as many anomalies as expected. The Scikit-learn library [28] provides two methods for defining "Gamma": "auto" and "scale".

Nu (default: 0.5): Allows us to control both the acceptable amount training errors and the number of support vectors derived from the training set. Higher "Nu" values mean that more training data points may be miss-classified in exchange for a larger number of them being used as support vectors. Smaller values of "Nu" mean less classification errors may occur with the training data but the minimum number of support vectors to use will also be lower.

Tolerance (default: $1\mathrm{e}{-3}$): Controls the algorithm's stopping criterion by defining the minimum gain each iteration must achieve to deem the algorithm worth continue running. The smaller the value of "Tolerance" the longer the algorithm will tend to run and approximate the optimal solution.

Standard Scaling (default: No): We will present a variation where a pre-processing scaling step was applied to the features fed into the anomaly detection technique, a procedure which has been found to improve OCSVMs' performance in the literature [29]. When this step was applied it standardized each features' samples x by calculating their standard score z. The formula by which z is derived consists of: $z = \frac{x-u}{s}$ where x is a sample of a given feature, u is the mean of that feature's samples and s is their standard deviation.

As a benchmark for this version of the system we chose to run the model with all default parameters. As was the case for every variant reliant on OCSVM, we opted for training and testing the model 10 times per configuration and taking the mean of TP and FP rates as the final result. Each time the process was as follows: i) of the 250 unwatermarked samples available per stream, 90% was taken at random and used for training the model; ii) the model was tested with the remaining 10% in order to determine the FP rate and iii) the model was tested with each of the 5 PCRs' 50 data points to determine the TP rate.

In the results for this benchmark we saw TP rates ranging from 74% to 92% and FP rates ranging from 47.2% to 53.1% (a considerable increase compared to the supervised learning benchmark of Section IV-C).

Besides the large FP rates we found that some of the tendencies previously observed did not seem to be present in this case: i) as was expected, the FP rates did not depend on the PCRs and ii) the TP rates, although showing similar values, did not seem to follow the trend of becoming greater as the PCR increased. This hints at the possibility of this approach allowing the system to be successful in configurations requiring less probes (granting more covertness).

When analysing the F1-score and P4 metrics, we found these indicated the Reply and Fetch streams as the best performing. The former achieved the highest scores in $2/5$ of PCR values while the latter did so for the remaining $3/5$.

E. Variants

After introducing a default parametrization of DissecTor's two watermark detection techniques, we will now present the results of experiments conducted with the aim of determining if the aforementioned results can be improved upon.

In the case of the supervised learning based version, we will present the results of experiments conducted with two alternatives to the C4.5 algorithm used in the benchmark: i) Naive Bayes [30] and ii) Random Forest [31]. As for the OCSVM-based version, we varied "Gamma", "Nu", "Tolerance" and "Standard Scaling" to study the impact of each parameter on the performance of the system.

In all cases, we ran experiments for all three streams (see Section III-B) and every PCR value (see Section IV-A.)

F. Supervised Learning Variants

Both alternative classifiers were used with the default parameters and with cross validation set to 10 folds. The datasets used in the variant experiments were exactly the same as in the benchmark which we have already covered.

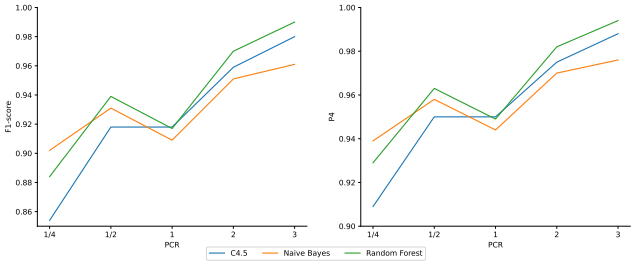

Figure 3. Best F1-score and P4 values of all three classifiers per PCR.

In the experiments conducted with the Naive Bayes classifier, the results showed the TP rate ranging from 80% to 98% and the FP rate ranging from 1.2% to 2.4%. We observed some of the tendencies that could be seen in the benchmark (see Section IV-C), namely: i) with two exceptions for $PCR = 1$, the TP rate increased with the PCR and; ii) the FP rate decreased with the PCR. We found the Reply and Joint streams performed similarly well for this classifier although analysis of both F1-score and P4 values led us to conclude the Joint stream is the best of the two, marginally outperforming the Reply stream in $4/5$ of PCRs.

In the experiments conducted with the Random Forest classifier, the results showed the TP rates ranging from 78% to 98% and the FP rates from 0% to 1.6%. As was the case with the other supervised learning algorithms, the TP rates tended to increase with the PCR and the FP rates tended to decrease with it.

In this particular case, the Reply stream achieved the best performance regarding both TP and FP rates, although it is noteworthy that a 0% FP rate was achieved for one of the other PCR-Stream combinations – Joint stream for $PCR = 3$. When looking at the F1-score and P4 values we confirmed the Reply stream was indeed the best suited for this particular system variant, as it outperformed the other streams in $4/5$ of PCR values.

When comparing the performance of the different classifiers we based our analysis on the F1-score and the P4 metrics. To do so, we determined the best values achieved by each classifier for every PCR. The best value for each PCR was chosen out of the three streams. Figure 3 presents our findings and allowed us to derive some conclusions: i) there is a clear degradation in performance for $PCR = 1$, most likely due to the fact that using a number of probes equal to the amount of baseline clients results in a watermark signature too similar to the regular traffic; ii) the Random Forest classifier (in green) outperforms the others in 3/5 of the PCR values while C4.5 (in blue) and Naive Bayes (in orange) performed the best in 1/5 of PCRs each; iii) although the figure does not explicitly specify it, when compiling the results it presents, we confirmed that the Reply stream was the best of the three for this analysis, followed by the Joint stream. The last conclusion is supported by the fact that the Reply stream achieved the best results in 60% of the cases, the Joint stream did so for 47% and the Fetch stream only for 13% (the total surpasses 100% due to some ties).

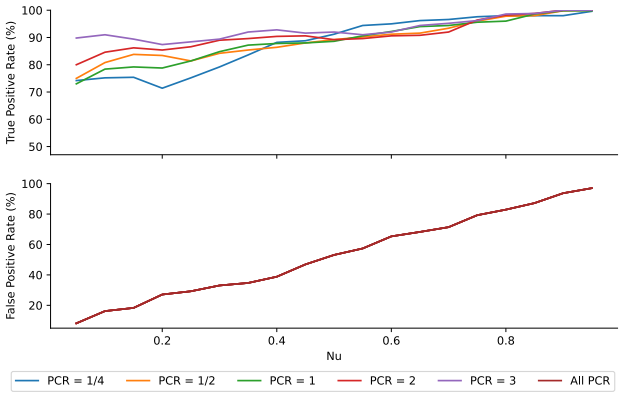

Figure 4. Impact of the "Nu" parameter on the true and false positive rates of DissecTor running with OCSVM in the otherwise benchmark configuration when analysing the Joint Stream.

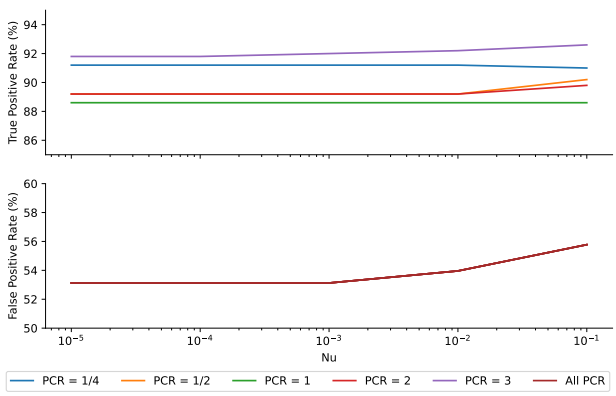

Figure 5. Impact of the "Tolerance" parameter on the true and false positive rates of DissecTor running with OCSVM in the otherwise benchmark configuration when analysing the Joint Stream.

G. Anomaly Detection variants

The results achieved for each anomaly detection variant of DissecTor were obtained in experiments where each parameter (see Section IV-D) was changed separately (keeping the others as their default values) in order to convey their impact on overall performance. In addition, we conducted experiments with every combination of parameters and will conclude this section by analyzing the "best" overall configurations, selected according to both F1-score and P4.

1) Standard Scaling: The results obtained when applying Standard Scaling showed TP rates ranging from 34.8% to 63.6% and FP rates ranging from 54.7% to 57%. This shows that, although FP rates were similar to those of the benchmark (see Section IV-D), FP rates suffered considerably. Our results suggest that, for this particular system, performing Standard Scaling is not beneficial.

2) Gamma: The default way of calculating the value of parameter "Gamma" ("scale") defined it as the inverse of the product of the number of features and their variance. In this variant we studied the impact of calculating "Gamma" with the alternative method "auto", that defines it as the inverse of the number of features.

The results of this experiment showed both TP and FP rates of 100%, independently of the PCR and the Stream being analysed. This indicates the value of "Gamma" was excessively large. In such cases, the importance given to each training set point is exaggerated and the model incurs in overfitting, classifying every data point outside the the training set as anomalies. We concluded this method of calculating "Gamma" was not suitable.

3) Nu: We studied the impact of this parameter by making it range from 0 to 0.95 in 0.05 increments. The remaining parameters were set to their dafaults.

Figure 4 shows the TP and FP rates achieved by the system for each PCR in the Joint Stream while varying "Nu". While the other streams are not depicted due to spacing restrictions, they behaved similarly and the conclusions we derived in regards to this parameter are shared by all. The top plot shows the TP rates obtained by every PCR as a function of "Nu" and the bottom plot depicts the FP rate also as a function of "Nu".

From the figure, we can conclude that an increase in "Nu" results in both higher TP and FP rates although in different proportions. Since the TP rates seem to not be affected as significantly by this parameter as the FP rates, we conclude the marginal gain in TP rate does not justify the relevant increase in FP rate and that DissecTor might benefit from smaller values of this parameter.

4) Tolerance: In our experiments, we ran the benchmark configuration, i.e., the one containing all default parameters and no standard scaling, but varied the value of "Tolerance" logarithmically between $1\mathrm{e}{-1}$ and $1\mathrm{e}{-5}$.

Figure 5 presents the TP and FP rates obtained in these experiments for the Joint Stream. The behaviour of the other streams was similar and, as such, for space related reasons, the other figures were omitted. In the figure, the top plot shows the TP rates of each PCR as a function of the "Tolerance" and the bottom plot depicts the FP rates. Note the use of a logarithmic scale in both plots.

From this figure, we can observe that the impact of this parameter is largely insignificant and is mainly perceivable when it reaches values $> 1\mathrm{e}{-3}$. For "Tolerance" $> 1\mathrm{e}{-3}$ we observed a slight increase in both TP and FP rates in almost every PCR-Stream pair, which might be a consequence of the model performing a smaller number of iterations to approximate the optimal solution and, therefore, ending up classifying a greater number of samples as anomalies.

Given the small impact increasing the "Tolerance" had on TP rates and the fact that it resulted in increases in already rather large FP rates, we concluded that smaller values of this parameter might be beneficial to the system.

5) Best configurations: In this section we discuss the results of using F1-score and P4 to select the best performing configurations for the OCSVM version of DissecTor for each of the PCR-stream pairs. The main purpose of this analysis is to allow us to confirm or debunk the conclusions derived when analysing all parameters individually. The results were taken from experiments conducted using all possible combinations of the parameters covered so far.

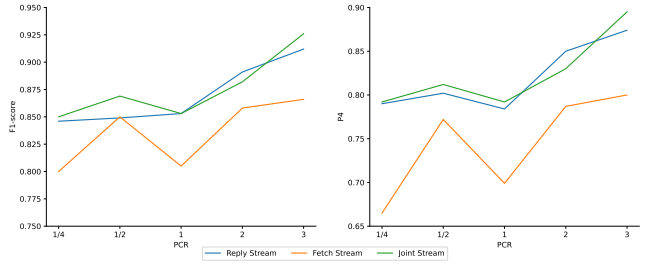

Figure 6. Best F1-score and P4 values of all three streams per PCR when using OCSVM.

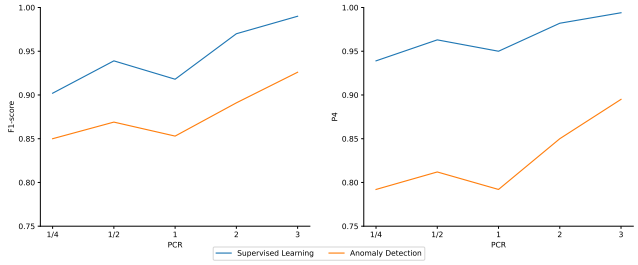

Figure 7. Comparison between the best F1-score and P4 values of both versions of DissecTor.

From the analysis of the results obtained with F1-score we concluded that our predictions were mostly correct since: i) no configurations used pre-processing with standard scaling; ii) all configurations except one used "scale" to calculate "Gamma"; iii) tolerance values were, with some exceptions in the Fetch stream, $\leq 1e{-}2$ and iv) again with the exception of the Fetch stream, most "Nu" values were smaller than the default (0.5), ranging from 0.05 to 0.45.

In general the TP rates of these configurations were extremely high (from 79.8% to 100%) even though some of the FP rates were so as well (particularly in the Fetch stream), ranging all the way from 8.2% to 100%. The existence of these discrepancies, and that of large FP rates among the "best" results, is a consequence of the F1-score metric as it prioritises large TP values over small FP ones. Thus, configurations that attained large TP rates were deemed as the best despite having FP rates we find unacceptable in a system that is to be deployed in the real world.

The best results according to the P4 metric further confirmed many of the tendencies pointed out throughout the evaluation of DissecTor. Although the F1-score results already did that to some extent, the results obtained this time around seem to support our observations more clearly: i) none of the configurations used the pre-processing step; ii) now, all configurations used "scale" to calculate "Gamma"; iii) most "Tolerance" values were still $\leq 1e{-}2$ and; iv) the overall values of "Nu" are not only lower than the default but lower overall when compared to the F1-score results, ranging from 0.05 to 0.25.

In general both TP and FP rates decreased when comparing to the previous metric. The TP rate now ranges from 66.2% to 89.8% while FP rates go from 5.4% to 27%. These values not only confirm that P4 results in more balanced results but also that many configurations of this version of DissecTor can achieve good performances in regards to TP and FP simultaneously.

Figure 6 shows the best results achieved with each stream for every PCR when evaluated for both F1-score (on the left) and P4 (on the right). Besides confirming the ineptitude of the Fetch stream (in orange) as a reliable source of data to perform watermark detection it shows both the Reply and Joint streams boast similar results. The fact that these were the best-performing streams across all versions of the system comes as no surprise as these are the ones containing the most traffic data. Furthermore, the figure shows how the insight regarding the performance degradation for $PCR = 1$ is valid across all versions of the system.

Another conclusion derived from the analysis of the best configurations (supported by Figure 6) resides in the fact that low PCR configurations may be successfully used when applying this technique as they achieved results that, not being the overall best, were not significantly worse than the ones where a larger amount of probes was used.

Finally, in Figure 7, we present a comparison of the best results of each watermark detection technique evaluated in this chapter for each PCR value. The figure shows that, although the best configurations of the OCSVM version of the system (in orange) still lack in comparison to their supervised learning counterparts (in blue), these are a viable alternative for scenarios where the latter is impractical.

It is important to note, however, that it is expected that the vast majority of the attacks performed by DissecTor resort to the anomaly detection technique, as the fact that the supervised learning approach relies on mock scenarios using extremely difficult to attain information should make them the rare exception and not the rule.

V. Conclusions

This work presented DissecTor, a distributed system whose goal is to allow LEAs to identify specific OSes' IP addresses, towards aiding digital investigation efforts. DissecTor assumes a model where multiple LEAs around the world cooperate and form coalitions so as to create a global network adversary. Taking advantage of this global adversary model, we proposed DissecTor, an active traffic correlation attack based on an acceleration-based watermarking technique. DissecTor makes use of different machine learning techniques to perform watermark detection and focuses on both accuracy and covertness. Our extensive evaluation of the system showed it could successfully be used to perform the attacks it was conceived for even in configurations that grant it covertness.

References

[1] Freedom of the Press Foundation, "SecureDrop," https://securedrop.org/, accessed: 2022-01-10.

[2] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.

[3] P. Winter and S. Lindskog, "How china is blocking tor," *arXiv preprint arXiv:1204.0447*, 2012.

[4] R. Hurley, S. Prusty, H. Soroush, R. J. Walls, J. Albrecht, E. Cecchet, B. N. Levine, M. Liberatore, B. Lynn, and J. Wolak, "Measurement and analysis of child pornography trafficking on p2p networks," in *Proceedings of the International Conference on World Wide Web*, 2013.

[5] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on tor using deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1962–1976.

[6] P. Medeiros, "Distributed system for cooperative deanonymization of tor circuits," January 2021.

[7] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows." in *Proceedings of the Network and Distributed Systems Security Symposium*, 2009.

[8] A. Iacovazzi, S. Sarda, and Y. Elovici, "Inflow: Inverse network flow watermarking for detecting hidden servers," in *Proceedings of the IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 747–755.

[9] F. Rezaei and A. Houmansadr, "Finn: Fingerprinting network flows using neural networks," in *Proceedings of the Annual Computer Security Applications Conference*, 2021, pp. 1011–1024.

[10] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on tor using long paths." in *Proceedings of the USENIX Security Symposium*, 2009, pp. 33–50.

[11] S. Chakravarty, A. Stavrou, and A. D. Keromytis, "Traffic analysis against low-latency anonymity networks using available bandwidth estimation," in *European symposium on research in computer security*. Springer, 2010, pp. 249–267.

[12] X. Wang, J. Luo, M. Yang, and Z. Ling, "A potential http-based application-level attack against tor," *Future Generation Computer Systems*, vol. 27, no. 1, pp. 67–77, 2011.

[13] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill, "Sleepy watermark tracing: An active network-based intrusion response framework," in *Proceedings of the IFIP International Information Security Conference*. Springer, 2001, pp. 369–384.

[14] A. Iacovazzi and Y. Elovici, "Network flow watermarking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 512–530, 2016.

[15] A. Houmansadr and N. Borisov, "Swirl: A scalable watermark to detect correlated network flows." in *Proceedings of the NDSS Symposium*, 2011.

[16] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "Dsss-based flow marking technique for invisible traceback," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 2007, pp. 18–32.

[17] A. Houmansadr and N. Borisov, "Botmosaic: Collaborative network watermark for the detection of irc-based botnets," *Journal of Systems and Software*, vol. 86, no. 3, pp. 707–715, 2013.

[18] Z. Ling, X. Fu, W. Jia, W. Yu, D. Xuan, and J. Luo, "Novel packet size-based covert channel attacks against anonymizer," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2411–2426, 2012.

[19] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 20–29.

[20] A. Iacovazzi, D. Frassinelli, and Y. Elovici, "The duster attack: Tor onion service attribution based on flow watermarking with track hiding," in *Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 213–225.

[21] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[22] R. Zhang, S. Zhang, S. Muthuraman, and J. Jiang, "One class support vector machine for anomaly detection in the communication network performance data," in *Proceedings of the 5th conference on Applied electromagnetics, wireless and optical communications*. Citeseer, 2007, pp. 31–37.

[23] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[24] Y. Sasaki *et al.*, "The truth of the f-measure," *Teach tutor mater*, vol. 1, no. 5, pp. 1–5, 2007.

[25] M. Sitarz, "Extending f1 metric, probabilistic approach," *arXiv preprint arXiv:2210.11997*, 2022.

[26] F. Eibe, M. A. Hall, and I. H. Witten, "The weka workbench. online appendix for data mining: practical machine learning tools and techniques," in *Morgan Kaufmann*. Morgan Kaufmann Publishers, 2016.

[27] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] M. Zhang, B. Xu, and D. Wang, "An anomaly detection model for network intrusions using one-class svm and scaling strategy," in *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2015, pp. 267–278.

[30] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 338–345.

[31] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.